

PATENT ABSTRACTS OF JAPAN

(11) Publication number: **09160949 A**

(43) Date of publication of application: 20 . 06 . 97

(51) Int. Cl.

G06F 17/50
G06F 9/06
G06F 11/28

(21) Application number: 07318688

(22) Date of filing: 07 . 12 . 95

(71) Applicant: **HITACHI LTD**

(72) Inventor: SUZUKI TAKASHI
UCHIBE KONAGI
SHONAI TORU

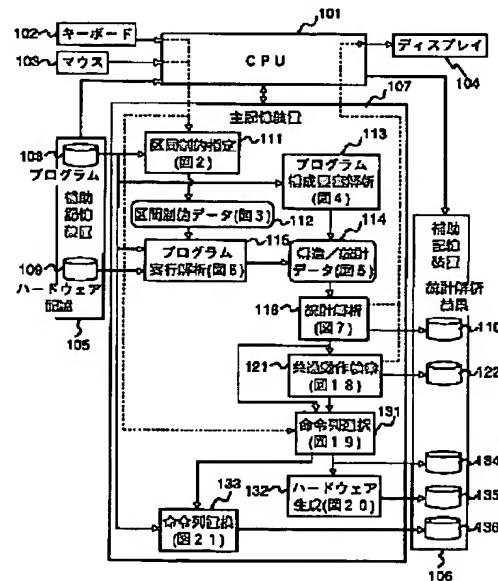
(54) DESIGN SUPPORTING METHOD IN MIXED SYSTEM OF HARDWARE AND SOFTWARE

COPYRIGHT: (C)1997,JPO

(57) Abstract:

PROBLEM TO BE SOLVED: To support the selection of a part to be made into a hardware in the functions of the control program of a microcomputer.

SOLUTION: The source program of incorporated programs 108 is displayed on a display 104, and the simulation starting point and the terminating point in the program and the processing time to be satisfied between the two points are made to be designated by a user (111). The object program analysis of the incorporated programs 108 is performed and the segments composing the function and the function as components and the mutual relation are extracted (113). The operations of the object program are analyzed, the processing time and the number of times of processing are recorded for every component executed till passing the terminating point after passing the starting point. These components and the processing time and the number of times of processing, etc., which are recorded on the components, are displayed (115). The user can select a part to be made into the hardware by a component unit.



(19)日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開平9-160949

(43)公開日 平成9年(1997)6月20日

(51) Int.Cl. ⁶	識別記号	序内整理番号	F I	技術表示箇所
G 06 F 17/50			G 06 F 15/60	6 5 4 M
9/06	5 3 0		9/06	5 3 0 U
11/28	3 4 0	7313-5B	11/28	3 4 0 C
			15/60	6 3 0

審査請求 未請求 請求項の数19 O L (全 20 頁)

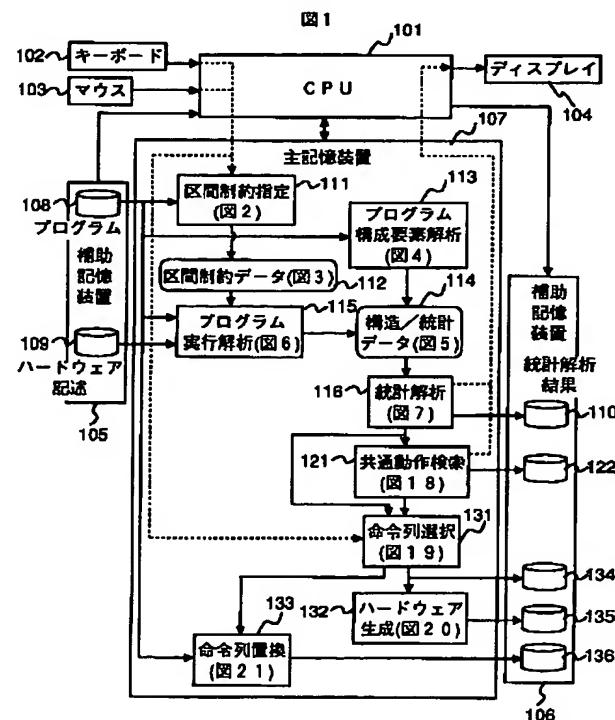
(21)出願番号 特願平7-318688
(22)出願日 平成7年(1995)12月7日

(71)出願人 000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目6番地
(72)発明者 鈴木 敬
東京都国分寺市東恋ヶ窪1丁目280番地
株式会社日立製作所中央研究所内
(72)発明者 内部 こなぎ
東京都国分寺市東恋ヶ窪1丁目280番地
株式会社日立製作所中央研究所内
(72)発明者 庄内 亨
東京都国分寺市東恋ヶ窪1丁目280番地
株式会社日立製作所中央研究所内
(74)代理人 弁理士 薄田 利幸

(54)【発明の名称】 ハードウェアとソフトウェアの混在システムの設計支援方法

(57)【要約】

【課題】マイクロコンピュータの制御プログラムの機能の内、ハードウェア化すべき部分の選択を支援する。
【解決手段】組み込みプログラム108中のソースプログラムをディスプレイ104に表示し、その中のシミュレーション開始点と終了点とそれら2点間の満たすべき処理時間とをユーザーに指定させ(111)、組み込みプログラム108中のオブジェクトプログラム解析してその構成要素としての関数と関数を構成するセグメントとおよびそれらの相互の関係を抽出し(113)、オブジェクトプログラムの動作を解析して、開始点を通過してから終了点を通過するまでに実行された構成要素毎に、処理時間と処理回数等を記録し、これらの構成要素と、それらに関して記録された処理時間、処理回数等を表示し(115)、ユーザーに、ハードウェア化すべき部分を構成要素単位に選択可能にする。



【特許請求の範囲】

【請求項1】プロセッサと、それに接続された少なくとも一つの関連回路とを有する回路を制御するためのオブジェクトプログラムの内、他の関連回路によりその機能を実行させるべき部分の選択を、計算機と、入力装置と、出力装置とを有する計算機システムを用いて支援するハードウェアとソフトウェアの混在システムの設計支援方法であって、

そのオブジェクトプログラムを構成する複数の構成要素とそれらの間の関係を示す構成要素間関係を判別し、

ユーザにより選択された、動作解析すべき区間を特定するための区間指定情報を上記入力装置を用いて入力し、該プロセッサと該関連回路を制御するときの上記オブジェクトプログラムの動作を、上記プロセッサの仕様と上記少なくとも一つの関連回路を記述するハードウェア記述とに基づいてシミュレーションし、

上記シミュレーションの間に、該判別された複数の構成要素の内、該入力された区間指定情報で特定される動作解析すべき区間に属する一部の複数の構成要素の各々の処理時間に関連する情報を収集し、

上記一部の複数の構成要素の各々を識別する情報とそれに対して収集された処理時間に関連する情報をと、該オブジェクトプログラムの内、ハードウェア化すべき部分の、該ユーザによる選択を支援するための情報として上記表示装置に表示するハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項2】上記オブジェクトプログラムを構成する該複数の構成要素の判別は、上記オブジェクトプログラムに対するソースプログラムを構成する複数の構成要素の一つにそれぞれ対応するように相対的に大きな複数の構成要素を選択し、該選択された相対的に大きな複数の構成要素の各々を構成する相対的に小さな複数の構成要素を選択し、上記選択された相対的に大きな複数の構成要素と上記選択された相対的に小さな複数の構成要素とをオブジェクトプログラムを構成する複数の構成要素として判別するステップを有し、

上記収集は、上記動作解析すべき区間に属する複数の相対的に大きな構成要素の各々に関する処理時間に関連する情報をと、それとの相対的に大きな構成要素に属する相対的に小さな複数の構成要素の各々についての処理時間に関する情報をとを収集するステップを含み、

上記表示は、上記動作解析すべき区間に属する上記判別された相対的に大きな複数の構成要素の各々およびそれとの相対的に大きな複数の構成要素に属する相対的に小さな複数の構成要素の各々に関して収集された処理時間に関連する情報を表示するステップを有する請求項1記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項3】上記オブジェクトプログラムは、上記オブジェクトプログラムに対するソースプログラムを変換し

て得られるアセンブラプログラムをさらに変換して得られたものであり、

上記オブジェクトプログラムの複数の構成要素と構成要素間関係の判別は、

上記アセンブラプログラムを解析して、アセンブラプログラムを構成する複数の構成要素とそれらの構成要素の間の関係を指示する構成要素間関係を判別し、

上記アセンブラプログラムを構成する複数の構成要素とそれらの構成要素の間の関係を指示する構成要素間関係

10 に基づいて、上記オブジェクトプログラムの対応する複数の構成要素と構成要素間関係を判別するステップを有する請求項1記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項4】上記オブジェクトプログラムの構成要素と構成要素間関係の判別は、上記オブジェクトプログラムを解析して行われる請求項1記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項5】上記区間指定情報の入力は、上記オブジェクトプログラムに対するソースプログラム

20 を上記表示装置に表示し、表示されたソースプログラム上の、ユーザにより選択された動作解析すべき区間の開始点と終了点とを指定する区間指定情報を上記入力装置を用いて入力するステップを有する請求項1記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項6】上記選択においては、

前記ソースプログラム上の上記開始点および上記終了点にそれぞれ対応する、上記オブジェクトプログラム上の開始点と終了点を、オブジェクト上の動作解析すべき区間を規定する情報として検出し、

30 該検出された開始点と終了点および該判別された構成要素間関係とに基づいて、該オブジェクトプログラム上の開始点を通過してから該オブジェクトプログラム上の終了点に至るまでの間に実行される一部の構成要素を、上記情報を収集する一部の複数の構成要素として選択するステップをさらに有する請求項5記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項7】上記オブジェクトプログラムを、該ソースプログラムから生成する時に、上記ソースプログラム内のコードと上記オブジェクトプログラム内の対応するコードとを関連づけるプログラム間関係情報を生成するステップをさらに有し、

上記検出においては、

上記生成されたプログラム間関係情報に基づいて、ソースプログラム上の上記開始点および上記終了点に対応する、上記オブジェクトプログラム上の開始点と終了点を検出するステップをさらに有する請求項6記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項8】上記動作解析すべき区間に属する該一部の複数の構成要素中の複数箇所に含まれる共通動作部分を

検索し、

該共通動作部分をそれぞれ含む複数の構成要素と、該共通動作部分のデータフローグラフとを該表示装置に表示するステップをさらに有する請求項1記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項9】プロセッサと、それに接続された少なくとも一つの関連回路とを有する回路を制御するためのオブジェクトプログラムの内、他の関連回路によりその機能を実行させるべき部分の選択を、計算機と、入力装置と、出力装置とを有する計算機システムを用いて支援するハードウェアとソフトウェアの混在システムの設計支援方法であって、

そのオブジェクトプログラムを構成する複数の構成要素とそれらの間の関係を示す構成要素間関係を判別し、ユーザにより選択された、複数の動作解析すべき区間の一つをそれぞれ特定するための複数の区間指定情報と各動作解析すべき区間が満たすべき、ユーザが指定した処理時間条件に関する情報とを上記入力装置を用いて入力し、

該プロセッサと該関連回路を制御するときの上記オブジェクトプログラムの動作を、上記プロセッサの仕様と上記少なくとも一つの関連回路を記述するハードウェア記述とに基づいてシミュレーションし、

上記シミュレーションの間に、該入力された複数の区間指定情報で特定される複数の動作解析すべき区間の各々の処理時間に関する情報と、各動作解析すべき区間に属すると判別された該一部の複数の構成要素の各々の処理時間に関する情報を収集し、

該複数の動作解析すべき区間の各々に対して収集された処理時間に関する情報とその区間にに対して入力された処理時間条件に関する情報に基づいて、上記複数の動作解析すべき区間の内、それぞれの区間にに対して指定された処理時間条件を満たさない一つまたは複数の動作解析すべき区間を検出し、

該検出された一つまたは複数の動作解析すべき区間の各々に対して検出された上記一部の複数の構成要素の各々を識別する情報と、それぞれの構成要素に対して収集された処理時間に関する情報とを、該オブジェクトプログラムの内、ハードウェア化すべき部分の、該ユーザによる選択を支援するための情報として上記表示装置に表示するハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項10】上記オブジェクトプログラムを構成する該複数の構成要素の判別は、上記オブジェクトプログラムに対するソースプログラムを構成する複数の構成要素の一つにそれぞれ対応するように相対的に大きな複数の構成要素を選択し、該選択された相対的に大きな複数の構成要素の各々を構成する相対的に小さな複数の構成要素を選択し、上記選択された相対的に大きな複数の構成要素と上記選択された相対的に小さな複数の構成要素と

をオブジェクトプログラムを構成する複数の構成要素として判別するステップを有し、

上記収集は、上記動作解析すべき区間に属する複数の相対的に大きな構成要素の各々に関する処理時間に関する情報と、それぞれの相対的に大きな構成要素に属する相対的に小さな複数の構成要素の各々についての処理時間に関する情報を収集するステップを含み、

上記表示は、上記動作解析すべき区間に属する上記判別された相対的に大きな複数の構成要素の各々およびそれ

10 ぞれの相対的に大きな複数の構成要素に属する相対的に小さな複数の構成要素の各々に関して収集された処理時間に関する情報を表示するステップを有する請求項1記載のハードウェアとソフトウェアの混在システムの設計支援方法。請求項9記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項11】上記オブジェクトプログラムは、上記オブジェクトプログラムに対するソースプログラムを変換して得られるアセンブラプログラムをさらに変換して得られたものであり、

20 上記オブジェクトプログラムの複数の構成要素と構成要素間関係の判別は、

上記アセンブラプログラムを解析して、アセンブラプログラムを構成する複数の構成要素とそれらの構成要素の間の関係を指示する構成要素間関係を判別し、

上記アセンブラプログラムを構成する複数の構成要素とそれらの構成要素の間の関係を指示する構成要素間関係に基づいて、上記オブジェクトプログラムの対応する複数の構成要素と構成要素間関係を判別するステップを有する請求項9記載のハードウェアとソフトウェアの混在システムの設計支援方法。

30 【請求項12】上記オブジェクトプログラムの構成要素と構成要素間関係の判別は、上記オブジェクトプログラムを解析して行われる請求項9記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項13】各区間指定情報の入力は、上記オブジェクトプログラムに対するソースプログラムを上記表示装置に表示し、

表示されたソースプログラム上の、ユーザにより選択された動作解析すべき区間の開始点と終了点とを指定する区間指定情報を上記入力装置を用いて入力するステップを有する請求項9記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項14】上記選択においては、

前記ソースプログラム上の一つの動作解析すべき区間を表す開始点と終了点に対応する上記オブジェクトプログラム上の開始点と終了点を、オブジェクト上の一つの動作解析すべき区間を規定する情報として検出し、

該検出された上記オブジェクト上の開始点と終了点および該判別された構成要素間関係に基づいて、該オブジェクトプログラム上の開始点を通過してから該オブジェ

クトプログラム上の終了点に至るまでの間に実行される一部の構成要素を、上記情報を収集する一部の複数の構成要素として選択するステップをさらに有する請求項13記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項15】上記オブジェクトプログラムを、該ソースプログラムから生成する時に、上記ソースプログラム内のコードと上記オブジェクトプログラム内の対応するコードとを関連づけるプログラム間関係情報を生成するステップをさらに有し、

上記検出においては、

上記生成されたプログラム間関係情報に基づいて、各動作解析すべき区間にに対する、前記ソースプログラム上の開始点と終了点に対応する、上記オブジェクトプログラム上の開始点と終了点を検出するステップを有する請求項14記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項16】上記複数の動作解析すべき区間に属する複数の構成要素中の複数箇所に含まれる共通動作部分を検索し、

該共通動作部分をそれぞれ含む複数の構成要素と、該共通動作部分のデータフローフラフとを該表示装置に表示するステップをさらに有する請求項9記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項17】プロセッサと、それに接続された少なくとも一つの関連回路とを有する回路を制御するためのオブジェクトプログラムの内、他の関連回路によりその機能を実行させるべき部分の選択を、計算機と、入力装置と、出力装置とを有する計算機システムを用いて支援するハードウェアとソフトウェアの混在システムの設計支援方法であって、

上記オブジェクトプログラムに対応するソースプログラムを変換して得られるアセンブラプログラムを上記オブジェクトプログラムに変換し、

該アセンブラプログラムに基づいて、該アセンブラプログラムを構成する複数の構成要素とそれらの間の関係を示す構成要素間関係を検出し、

該アセンブラプログラムを構成する複数の構成要素とそれらの間の関係を示す構成要素間関係に基づいて、該オブジェクトプログラムを構成する複数の構成要素とそれらの間の関係を示す構成要素間関係を判別し、

該プロセッサと該関連回路を制御するときの上記オブジェクトプログラムの動作を、上記プロセッサの仕様と上記少なくとも一つの関連回路を記述するハードウェア記述とに基づいてシミュレーションし、

上記シミュレーションの間に、上記判別された上記オブジェクトプログラムを構成する該複数の構成要素の内、少なくとも一部の複数の構成要素の各々の処理時間に関連する情報を収集し、

上記少なくとも一部の複数の構成要素の各々を識別する

ための情報とそれぞれに対して収集された処理時間に関連する情報を、上記オブジェクトプログラム中のハードウェア化すべきプログラム部分のユーザによる選択を支援する情報として上記表示装置に出力し、

上記アセンブラプログラムの複数の構成要素が識別可能なように、上記アセンブラプログラムを表示し、

上記表示されたアセンブラプログラムの内、ユーザにより選択されたハードウェア化すべきプログラム部分を指定する情報を該入力装置より入力し、

10 該入力された指定情報に従って、上記アセンブラプログラムの上記プログラム部分に代えて、他の関連回路を使用するように該アセンブラプログラムを更新し、該更新後のアセンブラプログラムに基づいて、上記変換から更新までを繰り返し、

その繰り返し時における上記更新後のシミュレーションの実行は、上記プロセッサの仕様と、上記少なくとも一つの関連回路を記述するハードウェア記述と、上記他の関連回路を記述する他のハードウェア記述とに基づいて実行されるハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項18】上記他のハードウェア記述を、該入力された識別情報指定される一つの構成要素の内容に基づいて自動的に生成するステップをさらに有する請求項17記載のハードウェアとソフトウェアの混在システムの設計支援方法。

【請求項19】上記他のハードウェア記述を該ユーザが指定するステップをさらに有する請求項17記載のハードウェアとソフトウェアの混在システムの設計支援方法。

30 【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、与えられた仕様のマイクロプロセッサを利用するシステムの設計にあたり、そのシステムが有すべき機能を、周辺回路により実現すべき一部の機能と、そのマイクロプロセッサの制御プログラムとで実現すべき機能とに分割するのを支援する方法に関する。

【0002】

【従来の技術】マイクロプロセッサを利用したシステム用のLSIの設計では、そのシステムが有すべき多数の機能の各々をハードウェアで実行させるかソフトウェアで実行させるかの選択が重要である。これは、ある機能をソフトウェアで実行させる方がLSIのコストが小さくて済み、修正も容易であるが、その機能をハードウェアで実行させる方が処理速度は速いというトレードオフが存在するためである。システムが有すべき多数の機能を、ソフトウェアで実現すべき機能とハードウェアで実現すべき機能とに区分することは、機能分割とも呼ばれる。このようなハードウェアとソフトウェアの機能分割における最も重要な基準はシステムの性能（処理速度）

である。しかし、性能以外にも、ハードウェアとソフトウェア間の転送データ量、LSIの面積（ハードウェアの面積とプログラムを格納するROMの面積）、消費電力、仕様変更への容易性等、機能分割に当たり考慮すべき点がいろいろある。

【0003】従来は、人手か、簡単なツールを用いてプログラムを解析し、ハードウェア化する部分を選択していた。あるいは、設計者の経験を利用してハードウェア化する部分を決定し、ソフトウェアの設計を行う過程で性能的に不十分な機能をその都度ハードウェア化するという手法をとっていた。この場合、ハードウェア化すべき機能を人手で仮に選択した後、選択された機能をハードウェア化した状態でのシステム性能等の評価が必要であり、この評価の結果によっては、ハードウェア化すべき機能を再度選択し直す必要が生じるため、ハードウェア化すべき機能の最終決定までには多くの時間が掛かる。

【0004】したがって、このような機能分割のためにには、ソフトウェアで実現される機能とハードウェアで実現される機能に機能分割した状態で、処理速度その他の特性に関してシミュレーション出来ることが望ましい。さらに、ハードウェア化すべき部分をこのシミュレーションにより自動的に決定できることが望ましい。さらに、ハードウェア化することが望ましいと決定された機能を実現するハードウェアは、計算機により自動的に生成出来ることが望ましい。

【0005】現在、ハードウェア／ソフトウェア混在システムの性能のシミュレーションを支援する手段として、論理シミュレータ上でマイクロプロセッサのモデルを動作させ、ハードウェア部分とプログラムを同時にシミュレーションする技術がある。例えば、情報処理学会大47回全国大会（平成5年後期）、6-85頁（以下、第1の参考文献と呼ぶ）には、プログラムのデバッグ機能を持つマイクロプロセッサモデルを論理シミュレータ上で動作させる技術が紹介されている。しかし、これらのハードウェア／ソフトウェア混在システムのシミュレーションの方法は、ハードウェア／ソフトウェア混在システムの動作を検証する目的で作られているため、処理速度を計る手段に関しては、プログラムの内、ユーザ指定の一部分の処理時間を測定する機能がある程度である。

【0006】したがって、システムの設計に当たり、そのシステムが有すべき機能の性能は、LSIにある信号が入力されてから、そのLSIが処理を行い別の信号を出力するまでの処理時間等で定義される。ソフトウェア上では信号の入力と信号の出力が同じサブルーチン内で行われる訳ではなく、一般的には異なるサブルーチンで行われ、その間に通過するプログラム上の経路も入力データの値や、その時のLSIの状態により変化する。また、ハードウェア化されるべき処理はサブルーチン内の

一部の処理であったり複数のサブルーチンにまたがる処理であったりする。そのため、ハードウェア化すべき機能の候補（プログラムの一部）を探すためには測定部分を何度も変えて繰り返し測定する必要がある。

【0007】一方、シミュレーションだけでなく、機能分割あるいはハードウェア化の自動化も研究されている。か一方、例えば、IEEE, Design & Test of Computers, Volume 10, Number 4, 1993年12月、64頁～75頁（以下、第2の参考文献と呼ぶ）に示されるように、ハードウェアとソフトウェアの切り分けの自動化を目指した研究も報告されている。この論文ではC言語で記述したシステムの動作記述からコプロセッサ（ハードウェア）とプログラム（ソフトウェア）を自動的に生成するシステムが提案されている。これらの研究では、ソフトウェアの動作を解析し処理頻度の高い部分を選んでハードウェア部分とする処理を自動化している。しかし、自動化の際のハードウェア化部分の選択の基準が処理頻度の高い部分だけであり、前記のような他の観点での評価は行われていない。このため、システムの使用を実現するのに最適な機能分割が実現できない。この問題を軽減するために、この従来技術では、設計者が直接ハードウェア化すべき部分を指定可能になっているが、その場合も、指摘できるのが、C言語のレベルのプログラム内の部分であり、実際の機械語レベルの命令列ではないために、設計者が意図した細かい分割を指定できないという問題がある。また、自動的にハードウェア化するための技術は、現段階では、十分実用的とは言えず、ある機能を実現するために、このような自動的な方法で生成されるハードウェアは、人手でその機能をハードウェア化した場合に得られるハードウェアに比べて、集積度あるいは処理速度等で劣る場合がある。

【0008】

【発明が解決しようとする課題】上記第2の参考文献がめざす機能分割の自動化は、理想的であるが、現実にはその技術は、まだ実用段階には達していない。このような技術の状況では、設計者がハードウェア化すべき部分を選択するのを支援できる技術がより実用的である。そのようは支援のためには、以下のよう条件を満たすことが望ましい。

【0009】（1）ハードウェア化すべき部分は、オブジェクトプログラム上の構成要素を単位として設計者が指定出来ることが望ましい。そのためには、シミュレーション結果は、オブジェクトプログラム上の構成要素の単位で得られることが望ましい。この場合、設計者が動作解析すべきプログラム部分を比較的容易に指定できることが望ましい。そのためには、動作解析すべき範囲をソースプログラム上で指定できることが望ましい。

【0010】（2）ユーザが指定した複数の動作解析すべき区間にに関するシミュレーションを行い、その結果に

基づいて、ユーザ指定の処理条件に違反したか否かを判別し、違反した複数の区間があれば、それらをハードウェア化すべき部分の候補としてユーザに提示し、ユーザがそれらの区間の各々の内部の内、ハードウェア化すべき部分を選択するのを支援するシミュレーション方法が望ましい。

【0011】(3)さらに、シミュレーションの結果を利用して一部の機能をハードウェア化した後に、その結果を利用して新たにハードウェア化すべき部分を見つけるように、シミュレーションを簡単に繰り返すことが出来ることが望ましい。

【0012】その際、設計者が望めば、設計者が選んだハードウェア化すべきプログラム部分を設計者が人手でハードウェア化した後に、その結果を取り込んで、新たなシミュレーションを実行できるシミュレーション方法が望ましい。

【0013】したがって、本発明の目的は、動作解析すべきオブジェクトプログラムの各構成要素毎にシミュレーション結果を出力できる設計支援方法を提供することである。

【0014】本発明のより具体的な目的は、これらの動作解析すべき区間をユーザが指定でき、シミュレーション結果として、オブジェクトプログラムの各構成要素毎にシミュレーション結果を出力できる設計支援方法を提供することである。本発明のさらに具体的な目的は、動作解析すべき区間をユーザがソースプログラム上で指定でき、シミュレーション結果として、オブジェクトプログラムの各構成要素毎にシミュレーション結果を出力できる設計支援方法を提供することである。

【0015】本発明の他の目的は、ユーザ指定の複数の動作解析すべき区間にに対してシミュレーションを行い、その結果に基づいて、設計者が指定した処理時間等の条件を満たさない複数の動作解析すべき区間を自動的に判別し、これらをハードウェア化すべきプログラム部分の候補として設計者に提示できる設計支援方法を提供することである。

【0016】さらに、本発明の他の目的は、先行するシミュレーションの結果を利用して次のシミュレーションが実行できるように、シミュレーションを簡単に繰り返すことを支援する設計支援方法を提供することである。

【0017】

【課題を解決するための手段】上記本願発明の目的を達成するために、本願発明では、計算機と、入力装置と、出力装置とを有する計算機システムにより以下を実行させる。

【0018】プロセッサと、それに接続された少なくとも一つの関連回路とを有する回路を制御するためのオブジェクトプログラムを構成する複数の構成要素とそれらの間の関係を示す構成要素間関係を判別し、ユーザにより選択された動作解析すべき区間を特定するための区間

10 指定情報を入力装置を用いて入力し、該プロセッサとその関連回路を制御するときの上記オブジェクトプログラムの動作を、上記プロセッサの仕様と上記少なくとも一つの関連回路を記述するハードウェア記述とに基づいてシミュレーションし、上記シミュレーションの間に、該判別された複数の構成要素の内、該入力された区間指定情報で特定される動作解析すべき区間に属する一部の複数の構成要素の各々の実行時間の合計、実行頻度など、その構成要素の処理時間に関連する情報を収集し、上記一部の複数の構成要素の各々を識別する情報とそれに対して収集された処理時間に関連する情報を、該オブジェクトプログラムの内、ハードウェア化すべき部分の、該ユーザによる選択を支援するための情報として表示装置に表示する。

【0019】より具体的には、上記複数の構成要素は、上記オブジェクトプログラムに含まれる複数の関数と、該複数の関数の各々に含まれる複数のセグメントとを含み、上記収集では、上記動作解析すべき区間に属する複数の関数の各々に関する処理時間に関連する情報と、それぞれの関数に属する複数のセグメントの各々についての処理時間に関する情報を収集し、上記表示では、上記動作解析すべき区間に属する上記複数の関数の各々およびそれぞれの関数に属する複数のセグメントの各々に関して収集された処理時間に関連する情報を表示する。

【0020】さらに、より具体的には、上記区間指定情報の入力では、上記オブジェクトプログラムに対するソースプログラムを上記表示装置に表示し、表示されたソースプログラム上の、ユーザにより選択された動作解析すべき区間の開始点と終了点とを指定する区間指定情報を上記入力装置を用いて入力する。

【0021】さらに、本発明では以上のことをユーザが選択した複数の動作解析すべき区間に關して行い、かつ、それぞれの区間に關してユーザが指定した処理時間等の条件を入力装置から入力し、それらの区間がユーザが指定した条件を満たしているかをそれらの区間に關するシミュレーションの結果により判別し、条件を満たさない複数の区間を検出したときには、それらの区間の各々に属する複数の構成要素に関するシミュレーション結果を表示装置に表示し、それでもってユーザがハードウェア化すべき部分を選択するのを支援する。

【0022】さらに、以上のようなシミュレーション結果を利用して、ユーザがいざれかのハードウェア化すべき部分を選択した後に、さらに他のハードウェア化すべき部分を選択するように、後続のシミュレーションを繰り返し実行するのを支援するために、本願発明のシミュレーション支援方法は、アセンブラプログラムを使用して、次のように行われる。

【0023】まず構成要素の解析は、オブジェクトプログラムに対応するソースプログラムを変換して得られるアセンブラプログラムを上記オブジェクトプログラムに

変換し、該アセンブラプログラムに基づいて、該アセンブラプログラムを構成する複数の構成要素とそれらの間の関係を示す構成要素間関係を検出し、該アセンブラプログラムを構成する複数の構成要素とそれらの間の関係を示す構成要素間関係に基づいて、該オブジェクトプログラムを構成する複数の構成要素とそれらの間の関係を示す構成要素間関係を判別することにより行われる。

【0024】上記オブジェクトプログラムのシミュレーションの結果表示の後に、上記アセンブラプログラムの複数の構成要素が識別可能なように、上記アセンブラプログラムを表示し、上記表示されたアセンブラプログラムの内、ユーザにより選択されたハードウェア化すべきプログラム部分を指定する情報を入力装置より入力し、該入力された指定情報に従って、上記アセンブラプログラムの上記プログラム部分に代えて、他の関連回路を使用するように該アセンブラプログラムを更新し、該更新後のアセンブラプログラムに基づいて、上記変換から更新までを繰り返す。

【0025】

【発明の実施の形態】以下、本発明に係るハードウェア／ソフトウェア混在システムの設計支援方法を図面に示した実施の形態を参考してさらに詳細に説明する。

【0026】(システム構成) 図1はハードウェア／ソフトウェア混在システムの設計支援装置の構成を示す。101はCPU、102はキーボード、103はマウス等の入力機器、104はディスプレイ等の出力機器である。105、106は補助記憶装置であり、105は入力用のファイルを、106は出力用のファイルを表している。107はCPUに接続する主記憶装置で、本発明による設計支援装置の動作を決定するプログラムとデータを格納する。

【0027】入力用ファイルは、設計対象となるLSIのうち、LSI内のROMに組み込むプログラムを格納するファイル108、既設計の部分の論理回路や設計対象となるLSIの周囲の動作を表すハードウェア記述109からなる。また出力用ファイルは統計解析結果を格納するファイル110、共通動作検索結果を格納するファイル122、選択した命令列を格納するファイル134、ハードウェアの生成結果を格納するファイル135、生成したハードウェアに対応する命令を用いて組み込みプログラム108を書き換えたファイル136からなる。

【0028】主記憶装置107には図1に示す設計支援装置の処理プログラムおよびデータを格納する。処理プログラムとデータは、区間制約処理111、区間制約データ112、プログラム構成要素解析処理113、構造／統計データ114、プログラム実行解析処理115、統計解析処理116、共通動作検索処理121、命令列選択処理131、ハードウェア生成処理132、命令列置換処理133からなる。

【0029】区間制約処理111は、組み込みプログラム108を読み込み、組み込みプログラムの中の2点間のタイミング制約をユーザに指定させ区間制約データ112を作成する。プログラム構成要素解析処理113は、組み込みプログラム108を読み込み、オブジェクトプログラムの構造を解析する。具体的には、オブジェクトプログラムの構成要素を解析し、さらにそれらの構成要素の相互の関係を解析し、構造／統計データ114を作成する。する。

【0030】一般に、プログラムは、ルーチンと呼ぶあるまとまった処理を実行するプログラム部分により構成される。ルーチンはメインルーチンとサブルーチンに分類できる。サブルーチンは、手続きあるいは関数とも呼ばれる。オブジェクトプログラム上のこれらのルーチンは、ソースプログラム上の一つのルーチンに対応する。従って、オブジェクトプログラムの構造を解析するときのプログラム構成要素として、ルーチンを解析すれば、それらはソースプログラム上のルーチンに対応するので、これらの構成要素はユーザにとってソースプログラムと対比して理解できる、分かりやすい構造といえる。このため、本実施の形態では、オブジェクトプログラムのルーチンを構成要素として解析する。しかし、各ルーチンは一般に多くの処理を実行するので、オブジェクトプログラムをハードウェア化する場合には、ルーチンより小さな構成要素についてハードウェア化するか否かを決めることが望ましい。このため、本実施の形態では、各ルーチンを構成する相対的に小さな構成要素も解析する。

【0031】以下に説明するように、本実施の形態では、一般に関数と呼ばれる複数のルーチンにより構成されたソースプログラムを使用するので、対応するオブジェクトプログラムの相対的に大きな構成要素の一例として、オブジェクトプログラムを構成する関数を解析する。各関数より相対的に小さなプログラム構成要素の一例として、セグメントを解析する。セグメントは、関数を構成する機械語レベルの命令列で、本実施の形態では、その途中に外への分岐点も外からの飛び込む点も持たない命令列をセグメントと定義する。このようなセグメントは、分岐に関連する処理を含まないために、オブジェクトプログラムのハードウェア化の際に、このセグメントを単位にしてハードウェア化するかあるいはその一部をハードウェア化するのに適している。

【0032】構造／統計データ114はセグメントや関数間の関係を表すデータと各セグメントと各関数毎の処理時間に関する統計情報を格納する。プログラム実行解析処理115は組み込みプログラム108とハードウェア記述109を読み込み、設計対象とするLSIのハードウェアとソフトウェア双方を同時にシミュレーションすることによりハードウェアの影響も考慮したソフトウェア(組み込みプログラム)の実行時の統計情報の採取

を行う。統計情報はセグメントや関数毎に構造／統計データ114に格納する。統計解析処理116はプログラム実行解析処理115の終了後に構造／統計データ114を解析して組み込みプログラムの処理時間に関する統計情報を計算しディスプレイ104と統計解析結果ファイル110に出力する。

【0033】共通動作検索処理121は、複数のセグメントに存在する類似の動作をする命令列を検索する処理である。ここで類似の動作とは、例えば、命令の実行順序が異なっていたり、計算時に使用するレジスタが異なっているが、同じ計算を行う場合を言う。検索結果はディスプレイ104とファイル122に出力する。

【0034】命令列選択処理131は、統計解析結果116、あるいは共通動作検索処理121により出力されたセグメント、あるいは共通動作命令列の中からハードウェア化する対象の命令列を選択する処理である。選択結果はファイル134に出力するとともに、ハードウェア生成処理132と命令列置換処理133に渡される。

【0035】ハードウェア生成処理132は、命令列選択処理131で選択されたハードウェア化対象の命令列を解析してその命令列と同じ動作を行うハードウェアの構成を求め、そのハードウェア記述をファイル135に出力する。命令列置換処理133は、組み込みプログラム108を読み込み、プログラム中に存在する命令列選択処理131で選択された命令列をハードウェア生成処理132で生成されたハードウェアを起動する命令列に置き換える処理である。置き換えた結果できたプログラムはファイル136に出力する。

【0036】以上の処理により組み込みプログラムの性能を解析し、与えた処理時間制約を満たさない部分に関してハードウェア化すべき部分を探索することができる。またハードウェア化すべき命令列からハードウェアを生成するとともに、元の組み込みプログラム中のハードウェア化対象命令列を生成したハードウェアを起動する命令列に置き換えることができる。さらに、置き換えた命令列と、生成したハードウェアを用いて再度プログラムの性能解析を繰り返すことにより、ハードウェア化の効果を確認し、生成したハードウェアの修正や、他のハードウェア化対象部分を検索することができる。

【0037】以下、以上の処理とそれに関係するデータの詳細を説明する。

【0038】(区間制約指定処理111) 図2は区間制約指定処理111の説明図である。区間制約指定では、ユーザーに動作解析すべき区間を指定する情報をキーボード102あるいはマウス103を用いて入力させる。さらにその区間が満たさなくてはならない処理時間の制約も指定させる。この処理を行うために、まずプログラム読み込み201により組み込みプログラム108を読み込む。

【0039】組み込みプログラム108は、高級言語で

記述したソースプログラムをコンパイルすることにより生成する。コンパイルは高級言語プログラムをアセンブラープログラムに変換する処理と、アセンブラープログラムをアセンブルし、機械語命令に対する命令コードと命令のアドレスからなるオブジェクトプログラムを生成する処理からなる。図8は組み込みプログラム108のファイル形式を説明する図である。図中で801はプログラムの格納されるアドレス、802は命令コード、803はラベル、804はコメント中に記録された高級言語のソースプログラム(805、807、809、811、813)とそのコンパイル結果であるアセンブリ言語(806、808、810、812)である。ソースプログラム解析202ではコメント中に記載されたソースプログラムの各文805とプログラムのアドレス801の関係を調べる。204のループはユーザーによる区間指定が終了するまで205～209の処理を繰り返す。開始点指定205ではユーザーによる開始点の指定を受け取る。指定はソースプログラムの文単位で行う。このように、ソースプログラム上でシミュレーションの開始点と終了点を指定する方法は、ユーザーにとって分かりやすい。このために、組み込みプログラム108をディスプレイ104に表示し、ユーザーにその表示された組み込みプログラム中のソースプログラム805上の、開始点の位置を区間指定情報として入力させる。開始点アドレス検索206ではオブジェクトプログラム解析処理203で解析した時に得られるプログラム間関係、すなわち、オブジェクトプログラムの各命令のアドレスと対応する高級言語プログラムの文との間の関係を基に、開始点に対応するオブジェクトプログラム上の命令の実際のアドレスを求める。終了点指定207と終了点アドレス検索208では同様にユーザー指定の終了点に対応するオブジェクトプログラム上の命令の実際のアドレスを求める。

【0040】処理時間制約指定209では205～208で指定した区間の処理時間が満たさなくてはならない時間の制約を受け取る。205～209で得た区間制約データは図3に示す区間制約データ112に格納する。図3において、1行が1つの区間制約データを表す。301は区間制約データの識別番号id、302は開始点アドレス、303は終了点アドレス、304は処理時間制約である。305はプログラム実行解析処理115と統計解析処理116で使用する区間毎の統計情報である。統計情報は複数の情報からなる。この詳細はプログラム実行解析処理115とともに説明する。

【0041】(プログラム構成要素解析処理113) 図4はプログラム構成要素解析処理113を説明する図である。プログラム構成要素解析処理113は、組み込みプログラム108を読み込み、プログラムの構造をセグメントと関数単位で解析し、構造／統計データ114を作成する。処理は402～408の処理を組み込みプロ

グラム108を1行ずつ読み込み(402)、解釈する。プログラムが関数の先頭であれば(403)、その関数の情報を構造／統計データ114に登録する。プログラム中の分岐点(ラベルが付加されているか、分岐命令がある場合)であれば(405)、その前に分岐点が出現したアドレスから現在の行のアドレスまでをセグメントとして登録する(406)。さらに、プログラム中に他の関数を呼び出すコード部分が存在すれば(407)、現在の関数の下位階層に呼ばれる関数があることを登録する(408)。図4の処理により、図8のプログラム例は806、808、810、812の4つのセグメントに分解される。

【0042】(構造／統計データ114)図5は構造／統計データ114を説明する図である。図は大きく関数の構造を表すFNstruct(501)、関数の情報を表すFN(506)、セグメント間の関係を表すSGstruct(512)、セグメントの情報を表すSG(525)からなる。FNstruct(501)は4つの要素からなる。fnp(502)はその関数の情報を表すFN(506)へのポインタ、sgl(503)はその関数を構成するセグメントの集合を表すリスト(518)へのポインタ、prnt(504)はその関数の上位階層の関数(その関数をコールする関数)のFNstructを指すポインタのリスト(519)へのポインタ、chd(505)はその関数の下位階層の関数のFNstructを指すポインタのリスト(520)へのポインタである。FN(506)は大きく5つの情報からなる。id(507)は関数の識別番号、FN名(508)は関数名、sa(509)は関数の開始アドレス、ea(510)は関数の終了アドレス、関数統計情報(511)はその関数の統計情報である。511はさらに複数の情報により構成される。この詳細はプログラム実行解析処理115とともに説明する。SGstruct(512)は4つの要素からなる。sgp(521)はセグメント情報SG(525)へのポインタ、pre(522)はこのセグメントの直前に実行されるセグメントのリスト541へのポインタ、suc(523)はこのセグメントの直後に実行されるセグメントのリスト542へのポインタ、fnp(524)はこのセグメントを含む関数(FNstruct)へのポインタである。セグメントの情報を表すSG(525)は大きく5つの情報からなる。id(526)はこのセグメントの識別番号、ラベル名(527)はこのセグメントの先頭アドレスにラベルがあればそのラベル名、sa(528)はそのセグメントの開始アドレス、ea(529)はそのセグメントの終了アドレス、セグメント統計情報(530)はそのセグメントの統計情報である。530はさらに複数の情報により構成される。この詳細はプログラム実行解析処理115とともに説明する。

【0043】(プログラム実行解析処理115)図6はプログラム実行解析処理115を説明する図である。プログラム実行解析処理115は組み込みプログラム108とハードウェア記述109により記述されるソフトウェアとハードウェア双方の動作を同時にシミュレーションし、ハードウェアの影響を考慮したソフトウェア(組み込みプログラム)の実行統計情報の採取を行う。ハードウェアとプログラムを同時にシミュレーションするために、通常の論理シミュレータ(601)上でハードウェア記述により定義される周辺回路602と組み込みプログラム108を格納するメモリのモデル603とマイクロプロセッサの動作を行うマイクロプロセッサモデル604を動作させる。そして、マイクロプロセッサモデルの実行する命令をトレースするプログラム実行トレース605によりプログラム実行統計情報を採取し、その結果は構造統計データ114に格納する。ここで採取する統計データは、セグメント毎と関数毎に、最大処理時間、最小処理時間、合計処理時間、処理回数、区間制約毎に最大処理時間、最小処理時間、合計処理時間、処理回数、制約値を超えた回数、処理中に通過したセグメントのリストである。また統計を採取する際にしようするデータとして各セグメント毎に開始時刻、終了時刻、経過時間、最終実行アドレス等の情報を、各区間制約毎に開始時刻、終了時刻、集計中を示すのフラグ等の情報を持つ。

【0044】図10はプログラム実行トレース処理605を詳しく説明する図である。プログラム実行トレースの処理は論理シミュレータ601上でのシミュレーションの各サイクル毎に起動される。処理はマイクロプロセッサモデル604上で実行されている命令のアドレスadとシミュレーション時刻ctを用いる。まず、現在の実行アドレスadを含むセグメント番号csを求める(1001)。次に、区間制約データ112に格納されているすべての区間制約tsnに関して1003～1006の処理を行う。現在の実行アドレスadが開始アドレスならば1004～1005を、終了アドレスなら1006を行う(1003)。開始アドレスなら現在参照している区間制約tsnに関して、統計データを集計中でなければ(1004)、その区間制約tsnの統計データの集計を開始する。一方、終了アドレスなら集計を開始してから現在までの時間を統計情報305を更新する。次に現在のセグメント番号csが、一回前にプログラム実行トレースを行った際のセグメント番号snとが等しくない場合、即ち、現時刻で新しいセグメントに入った場合に(1007)、現在集計中の区間制約tsaに関して(1008)、現在のセグメント番号csを統計情報305中の通過セグメントリストに追加する(1009)。さらに、現在のセグメントcsの統計情報30を更新する(1010)。この処理は図17で詳しく説明する。そして、現在のセグメントcsをsnとし

(1011)、最終シミュレーション時刻 X_{time} を現在の時刻 c_t とする (1012)。

【0045】図17は現在のセグメント c_s の統計情報 530の更新処理 1010を説明する図である。この処理も現在のセグメント番号 c_s が、一回前にプログラム実行トレースを行った際のセグメント番号 s_n とが等しくない場合とそうでない場合で処理が異なる (1701)。等しい場合は、セグメント c_s の最後に実行したアドレスを現在の実行アドレス a_d として記録する (1702)。等しくない場合は、セグメント c_s の実行開始時刻を現在のシミュレーション時刻 c_t として記録し (1703)、セグメント c_s が処理の一番最初のセグメントでない限り (1704)、今までのセグメント s_n の終了時刻を c_t とし (1705)、今までのセグメントの最終実行アドレスとセグメントの終了アドレスが等しい場合 (1706)、セグメント s_n の処理時間を計算し (1707)、それを基にセグメント s_n の統計情報を更新し (1708)、処理時間の計算に用いる経過時間をクリアする (1709)。一方、等しくない場合、即ち、割り込み等の要因で、セグメント途中で他のセグメントに処理が移った場合は、経過時間に今までの処理時間を加え保持する (1710)。

【0046】以上では、ソフトウェア中の一部の指定された区間に対して実行解析を行ったが、プログラム全体をミュレーション区間として指定したい場合には、シミュレーションの指定区間の開始点と終了点として、プログラムの先頭およびオブジェクトプログラムの命令を持つことがありえない命令アドレスを指定すればよい。

【0047】(統計解析処理 116) 統計解析処理 116はプログラム実行解析処理 115により求めた各セグメント毎の処理時間や実行回数などの値から、区間制約、セグメント、関数毎の処理頻度、実行回数などの統計値を計算し、出力する処理である。この出力では、計算結果をファイル 110として出力するとともに、ディスプレイ 104へその内容を表示する。

【0048】図7は統計解析処理 116を説明する図である。統計解析処理 116では、まず各セグメント s_g について (701)、全処理時間に対する s_g の処理時間割合と s_g の平均処理時間を計算 (702) し、各関数 f_n に関して (703)、 f_n 自体の合計処理時間、全処理時間に対する f_n の処理時間割合、 f_n の実行回数、 f_n の平均処理時間、 f_n 以下の関数の合計処理時間、 f_n 以下の関数の平均処理時間を計算 (704)。そして区間制約違反部分に関して処理時間に関連する情報の表示を行う (705)。この処理は図11で詳しく説明する。最後に区間制約、セグメント、関数毎の処理時間に関連する情報を統計解析結果ファイル 110に出力する。

【0049】図11は区間制約違反部分に関して処理時間に関連する情報の表示を行う処理 (705) を説明す

る図である。これは図12～図16に示すディスプレイ上に表示される情報がどのような手順により表示されるかを示す状態遷移図である。区間リスト表示 1101は、区間制約に違反した、即ち、実際の処理時間が区間制約としてあらかじめ指定された処理時間 304を超えた場合にその区間制約を表示する。1101から一つの区間制約を選択し (1104)、その区間を処理する際に通過したセグメントのリストを表示する (1105)。ここで、一つのセグメントを選択し (1108)、セグメントの中の命令列を表示できる (1109)、またはそのセグメントを含む関数、さらにその上位の関数といったプログラムの階層を表示できる (1110)、プログラムの階層表示で、一つのセグメントを選択すると、セグメントリスト表示 1105に戻り、一つの関数を選択すると関数のリストを表示できる (1113)。関数表示から関数を選択すると、その関数に含まれるセグメントのリスト (1105)、またはその関数を処理中に通過した区間を表示できる (1101)。

【0050】区間リスト表示 1101、セグメントリスト表示 1105、関数表示 1113は表形式の表示で、さらに2つの状態を含む。第一の状態は表のある列が選択された状態である (1103、1107、1115)。列が選択されると、選択された列の表示内容によりソーティングして再表示する。第二の状態は表のある行が選択された状態である (1104、1108、1116)。行が選択され、再度指示があると、表示が変わる。たとえば、状態 1104から状態 1108へ、状態 1108から状態 1112または状態 1104へ、状態 1116から状態 1112または状態 1108または 1104へと変わる。

【0051】図12～図16を用いてディスプレイ上に表示する情報の例を示す。これらの表や図は、ディスプレイ上のウインドウに個々に表示する。またウインドウの大きさはキーボードやマウスによる指示により変更できる。

【0052】図12は区間制約に違反した区間情報の表示例である。1201はタイトル部分で、この表のタイトル「区間制約違反」の他、表の各項目のタイトルを表示する。表の各項目は # (1202) が各行の番号、 t s_n # (1203) が区間制約データの識別番号 i_d (301)、start (1204) が開始点アドレス (302)、end (1205) が終了点アドレス (303)、spec (1206) が制約として与えられた処理時間 (304)、ave (1207) がその区間の平均の処理時間、cnt (1208) がその区間の実行回数、violate (1209) が実際の処理時間が制約時間を超えた回数である。

【0053】図12や他の類似の表形式の表示 (図13、図16、図19) に共通する機能として、行選択機能、列選択機能、表示範囲変更機能がある。行選択機能

は表の1つの行を選択する機能で、ディスプレイ104上に表示されるマウス103のカーソルを選択したい行に合わせ、マウスのボタンをクリックすることにより選択する。選択するとその行の表示の背景色と文字色を反転する。初期状態では最も上にある行が選択されている。列選択機能は表の各行の表示順序を変更する機能である。ウィンドのタイトル部分の各項目にマウスのカーソルを合わせ、マウスのボタンをクリックすることにより、その項目に関してソーティングを行い再表示を行う。選択した項目が数値であれば大きい順に、文字であればアルファベット順にソーティングする。表示範囲変更機能はスクロールバー(1210～1212)を用いて表の表示範囲を変更する。要素数が多い表では狭いウィンド上に多くの情報を表示できないため、一部分のみを表示する。その時、全体に対する表示箇所の位置が1212の位置で表わされる。1212にマウスのカーソルを合わせ、マウスのボタンを押したままカーソルを上下することにより表示範囲を上下に移動する。また1210、1211にマウスのカーソルを合わせ、マウスのボタンをクリックすることで、上下方向にそれぞれ1行分表示範囲をずらすことができる。図11に示すような他の表示へ状態を移動するには、選択した行の上にマウスのカーソルを合わせマウスのボタンを押し続けた際に画面上に現われる移動可能な表示状態のリストのうちの一つをマウスのカーソルで選択することにより行う。すでに目的のウィンドが画面上に表示されている場合は、行選択の結果、他のウィンドの表示内容も連動して変化する。またカーソルをそのウィンドに合わせることでそのウィンドの状態に移ることができる。なお、図12においては初期状態で1207の項目に関してソーティングして表示する。

【0054】図13はセグメントの実行統計の情報の表示例である。1301はタイトル部分で、この表のタイトル「セグメント実行統計」の他、表の各項目のタイトルを表示する。各項目は、#(1302)が各行の番号、sg#(1303)がセグメント番号(SGのid526)、label(1304)がセグメントの先頭アドレスのラベル(SGのlabel名527)、start(1305)はそのセグメントの開始アドレス(SGのsa528)、end(1306)は終了アドレス(SGのea529)、total(1307)はそのセグメントの合計処理時間、rate(1308)は全処理時間に対するそのセグメントの処理時間の割合、call(1309)はそのセグメントの実行回数、ave(1310)はそのセグメントの平均処理時間、min(1311)は最小処理時間、max(1312)は最大処理時間である。なお、図13においては初期状態で1307の項目に関してソーティングして表示する。

【0055】図14は図13の第1要素のセグメントの

例である。1401はタイトル部で、図13における#、sg#、start、endの情報を表示する。1402はアドレス部、1403はラベル部、1404は命令部である。命令部にはアセンブラーの命令列の他、アセンブラーに対応するソースプログラムをコメントの形で表示する。これらの命令列の情報は組み込みプログラムファイルを参照して表示する。

【0056】図15は関数とセグメントの階層を表わす。例えば、図13でlabelがL103の行を選択している場合、L103(1504)とそれを含む関数(1503)やその上位関数(1502)、あるいはL103と同じ関数に含まれるセグメント(1505～1507)、そして上位関数に含まれる関数(1508)とセグメント(1509)が表示される。関数名あるいはセグメント名を表示したボックスにマウスのカーソルを合わせ、マウスのボタンをクリックすると、選択した部分を中心とした表示に切り替わる。

【0057】図16は関数の実行統計の情報の表示例である。1601はタイトル部である。表の各項目は、#(1602)が各行の番号、fn#(1603)が関数の識別番号(FNのid関数507)、name(1604)が関数名(FNのFN名508)、start(1605)が開始アドレス(FNのsa509)、end(1606)が終了アドレス(FNのea510)、total(1607)がその関数自体の合計処理時間、rate(1608)が全処理時間に対するその関数の処理時間の割合、call(1609)がその関数の実行回数、ave(1610)がその関数平均処理時間、all(1611)がその関数以下の全処理とその関数の処理時間合計、arate(1612)がallの全処理時間に対する割合である。この表は初期状態でtotalの大きい順に表示する。

【0058】(共通動作検索処理121)この処理は、ハードウェア化すべき部分の候補として、プログラム中の異なる部分で実行される同一の処理の部分を検出し、ユーザに提示するために行う。図18は共通動作検索処理121を説明する図である。図18の1801では各セグメントsgについて1802～1805の処理を繰り返す。1802ではsg中の演算命令をノードとするデータフローグラフdfgを作成する。1803ではdfg中で出枝が2以上のノードがルートとするような木に分割し個々の木をクラスタとする。例えば、図22に示すデータフローグラフの場合、出枝が2以上のノード2206と2208、出枝が無い2210と2211をルートとするような木に分割し個々の木(2212、2213、2214、2215)をクラスタとする。1804では命令毎にあらかじめ決められた命令コストの総和とそのクラスタの実行回数(すなわちセグメントの実行回数)の積を求め、これをクラスタのコストCcとする。1805ではクラスタをその中の構造、演算命令の

種類により分類し、種類毎に識別番号 $i \cdot d$ を付加する。1806～1807では同一クラスタ種の2つのクラスタの組 i, j に関して、クラスタ i と j が同一の種類のクラスタを出力側または入力側に持つ場合に i と j を同一のグループとする処理を行う。1808～1809では1807で作成した各グループについて、1807と同様の処理を繰り返すことで、グループに属するクラスタを増やしていく。この処理により、共通のクラスタ構造を持つ極大のグループを構成できる。1810～1811では1809で拡大したグループについて、グループを構成するクラスタのノード数があらかじめ決められた値 N よりも小さいグループの登録を抹消する。1812では、各グループについて、グループを構成するクラスタのコストの総和が大きいグループから順にグループのデータフローグラフとグループを構成するクラスタを含むセグメント番号を整列して共通動作命令列情報を生成し、出力する。この出力では、この共通動作命令列情報をファイル122へ出力するとともに、ディスプレイ104に表示するように行う。

【0059】図19は1812の処理によりディスプレイ104に表示される共通動作命令列情報の表示例である。1901はタイトル部である。表の各項目は# (1902) が各行の番号、gr# (1903) が共通動作の命令列のグループ番号、cost (1904) がそのグループのコストの総和、#cl (1905) がそのグループを構成するクラスタの種類の数、#sg (1906) がそのグループの共通動作命令列を含むセグメントの種類の数である。この表はcostの大きい順に表示する。また、マウスのカーソルを任意の行(共通動作命令列グループ)に合わせ、マウスのボタンをクリックすると、図23に示す様な命令列のデータフローグラフを表示するウィンドを開くことができる。また同時に、もし図13に示すセグメント情報の表示が画面上にあれば、そのグループの共通動作命令列を含むセグメントに対応する行の表示の背景色と文字色を反転する。

【0060】図23はデータフローグラフの表示例であり、2301はデータ入力、2302はデータ出力、2303～2305は演算ノードを表わす。このグラフでは命令列中のデータ移動命令(メモリ読み出し、レジスタ間の移動)やフロー制御命令(分岐)は省略される。

【0061】(命令列選択処理131) 命令列選択処理131は、ハードウェア化の対象となるプログラムの一部としてユーザが選択した命令列を入力する。選択の対象は、共通動作命令列、セグメント内的一部の命令列、複数のセグメントにまたがる命令列である。ユーザは、統計解析処理116の処理結果あるいは共通動作検索処理121の処理結果をディスプレイ104上で見て、ハードウェア化すべきプログラム部分を選択し、選択した部分を、ディスプレイ104上で指示する。たとえば、統計解析処理116の結果に従って、このハードウェア

化する部分を選択するときには、図12あるいは図13等の、ディスプレイ104に表示される情報を見て、ハードウェア化すべき部分をこれらの画面上で指示する。また、共通動作検索処理121で検索された共通動作命令列またはその一部をハードウェア化すべき部分として選択する場合、図19上で一つの行を選択する。

【0062】(ハードウェア生成処理132) ハードウェア生成処理132は命令列選択処理131が選択した命令列について、ユーザより、その命令列を実行するハードウェアを自動生成することが指示された場合に行なわれ、ユーザ自身がその命令列を実行するハードウェアを設計する場合には、この処理は行われない。このハードウェア生成処理132で生成されるハードウェアが実用的と期待される場合には、ユーザは、この処理を起動すればよい。あるいは、テスト的にこの処理を起動し、この処理で生成されるハードウェアを評価した後、このハードウェアが実用的でないと判断したときには、この処理で生成されたハードウェアを破棄するようにしてもよい。以下では、ユーザが共通動作検索処理121で検索された共通動作命令列をハードウェア化すべき命令列として選択した場合を例に取り、ハードウェア生成処理132を説明する。

【0063】図20はハードウェア生成処理132の動作を説明する図である。まず、ユーザが選択した命令列を解析し、図23に示すようなデータフローグラフを作成する(2001)。次にデータフローグラフの各ノードにある演算命令に対応してあらかじめ定義されている演算器を調べる(2002)。そして、使用する演算器の情報とデータフローグラフの構造に合わせた演算器の接続情報を含んだハードウェア記述を生成する。この時、生成したハードウェアはマイクロプロセッサのメモリ空間に設定されたレジスタを入出力とし、メモリ空間の決められたアドレスを参照することにより起動する。生成は回路の雛型(HDL)をライブラリに用意し、これを実際のデータフローに対応して変更することにより行う。

【0064】例えば、図23に示すデータフローグラフに対して図24のような論理回路を生成するが、この図で2405、2407の入力レジスタが図23の入力2301に、2412の出力レジスタが出力2302に、2409の乗算器が2303に、2410の乗算器が2404に、2411の加算器が2405にそれぞれ対応する。実際には図25と図26に示すHDLが生成の対象である。2501部分は回路名称と入出力端子を定義している。この中で回路名称と、入力レジスタの書き込み制御線の本数(S TR0, S TR1)が設計対象により異なる。2502部分はライブラリファイルに定義されている各演算命令に対応した演算器とレジスタの定義である。これらは、設計対象のデータフローグラフのノードに応じて必要な演算器を選択する。2503は生成

回路中で用いる定数と信号線である。これらは、このハードウェアの起動命令で用いる割り当てアドレスとこのハードウェアの実行サイクル、使用する演算器とレジスタの数により決める。ハードウェアの実行サイクルは入力レジスタと出力レジスタ以外のレジスタの段数+1の値である。2601は図24の制御回路2404の動作である。これらは実行サイクル、割り当てアドレス部分の名称を設計対象により変更する。2602はレジスタと演算器の接続情報である。これらは図23のグラフに対応して生成する。ハードウェア生成処理では、図25、図26に示したHDLの他に、入出力レジスタの選択信号を作成するアドレスデコーダの回路を用意する。このアドレスデコーダもライブラリファイルにあらかじめ登録したHDL記述を用いる。このハードウェア生成には特開平4-175974に示した方法を用いることもできる。

【0065】なお、以上において、プログラム中の一部の命令列と等価な動作を行うハードウェアを人手で作成した場合は、そのハードウェア記述を既設計のハードウェア記述を格納したファイル109に追加する。ハードウェアの起動の方法が、すでに示した方法と同様の場合は、ハードウェア化した命令列をデータフローの形式で入力すると、後に述べる命令列置換処理133によるプログラムの修正が可能である。また、このような人手によるハードウェア記述を含むプログラムの修正も人手で行うことも可能である。追加したハードウェアの記述と、そのハードウェアを起動する命令列を含むプログラムが準備できれば、上に示した手順でハードウェア込みの性能解析が可能になる。この時、ソースプログラム上で指定した区間制約の箇所をソースプログラムレベルで保存しておくことにより、プログラムのサイズの変化に伴い、制約区間の実際のアドレスが変わっても、再度区間制約の指定を行わずに同じ制約箇所に関する情報を採取することができる。

【0066】(命令列置換処理133) 命令列置換処理133はハードウェア生成処理132の実行をユーザが指定した場合に行われ、ユーザが選択したハードウェア化すべき命令列として選択した命令列と、ハードウェア生成処理132が生成したハードウェアを起動するための命令列を用いて、組み込みプログラム108に記述された命令列をハードウェア生成処理132が生成したハードウェアを利用する命令列に自動的に置き換える。

【0067】この処理を図21を用いて説明する。2101では、命令列選択132が選択した命令列のデータフローグラフdfgを作成し、2103~2110の処理を、命令列の置き換えが必要な各セグメントsgについて繰り返す。2103ではsgのすべての命令をノードとするデータフローグラフdfg_sgを作成する。2104ではdfgとdfg_sgの対応する命令を検索する。2105ではdfg_sg上の対応演算命令の

演算対象のレジスタ、あるいはメモリ位置を指すインデックスレジスタの値をセットしている命令を検索する。2106では2105で求めたレジスタのセットの命令を、生成したハードウェアの入力レジスタへの書き込み命令に変更する。2107ではdfg_sg上の対応演算命令のうち、最後の演算命令とその結果が格納される出力レジスタの番号Nrを検索する。2108では最後の演算命令を、生成したハードウェアの起動命令と、生成したハードウェアの出力レジスタからレジスタNrへの書き込み命令に変更する。2109ではdfg_sg上の対応演算命令のうち、変更されていない残りのノードを削除する。2110ではdfg_sgのノードをトポロジカルオーダで(入力の依存順)アセンブリ命令に置き換えてファイル136出力する。

【0068】図9は命令列置換の結果を説明する図である。2701は図8に示したプログラムである。ここでは図23に示すデータフローグラフが表す共通動作命令列を置換の対象とする例を説明する。図23の共通命令列を含むL103から始まるセグメント(810)は、命令列置換の処理により2702のように置き換わる。図9に示す結果はこの形式でファイル136に格納される。命令列置換により命令数が変化するので従来のアドレスや命令コードはそのまま再利用することはできないが、ファイル136を再度アセンブルし、図8に示すようにアドレス、命令コードを含む形式に変換することにより、再度本実施の形態に示す方法を適用することができる。すなわち、置換後のプログラムファイル136をアセンブルし、ハードウェア生成処理132が生成したファイル135をハードウェア記述109に追加して、本実施の形態に示す方法を再度適用することで新しく生成したハードウェアを含めてシステム性能を評価し、ハードウェア化の効果を確認できるとともに、新たなハードウェア化箇所の候補を選択することができる。

【0069】以上の発明の実施の形態から分かるように、マイクロプロセッサとそのプログラムを組み込んだメモリ、および周辺回路を1チップに集積したLSIのようなハードウェアとソフトウェアが混在したシステムの設計に際し、ハードウェア部分の動作を含めて、ソフトウェアの処理時間と処理頻度を正確に測定し、指定区間の処理時間制約を満足しない箇所を調べ、その動作を抽出するので、ユーザがハードウェア化すべき部分を選択するのが容易になる。さらに、目標性能を満足するよう、少ない追加ハードウェア量で目的の動作を実現しようとする場合に、処理時間制約を満足しない指定区間に通過するプログラム部分の処理頻度を調べることにより、ユーザがハードウェアとして実現するプログラムの1部分の候補を探すことが容易にできるようになる。さらに、同じ動作を異なる命令順で行っているプログラム中の共通動作部分を自動的に調べて、ユーザに提示した場合には、プログラムの構成要素の一部あるいは複数

の構成要素にまたがった、ハードウェア化すべき範囲として、ユーザがこの共通動作部分を選択しやすくなる。選択したプログラム動作から自動的に生成したハードウェアモデルが実用的であると期待される場合には、ユーザの指示によりそのハードウェアモデルを自動的に生成し、元のプログラムを生成したハードウェアを利用するように自動的に書き換えることもできる。この書き換える後のプログラムの性能評価を再度おこなうことによりハードウェア化の効果を簡単に確認することが可能になる。さらに、この作業を繰り返すことにより、すでに生成したハードウェアの修正や他のハードウェア化対象部分を検索することができる。

【0070】<変形例>

(1) 前述の実施の形態で記載した技術は、マイクロプロセッサ以外のプロセッサ、たとえば、信号処理プロセッサ等を搭載した回路にも適用できる。さらに、プロセッサが搭載されているLSIの外部に位置する他のLSIに一部の周辺回路が搭載されている場合にも、適用できる。

【0071】(2) 前述の実施の形態では、区間制約の指定に際し、ソースプログラム上で指定する例を示した。これは、区間制約に関する統計情報は、機械語命令レベルで採取するが、ユーザにとって機械語命令レベルより、ソースプログラムで指示を行う方が容易であることによる。しかし、画面上で機械語命令のアドレスで区間を指定する、またはファイルに記述した機械語命令のアドレスで区間を指定することもできる。

【0072】(3) 前述の実施の形態では、組込みプログラムの例として機械語プログラム、アセンブラー、ソースプログラムが混在する形式のファイルを入力し、アセンブラーの記述をベースにして構成要素解析を実施する例を示したが、機械語プログラムと、各関数の開始アドレスや分岐先のアドレス等のコンパイル時の情報を元に同様の解析を行うこともできる。この場合、機械語プログラムを解析し、分岐命令の分岐先アドレスを特定することにより、セグメントへの分割を行う。分岐先アドレスがレジスタ値により指定されるような分岐命令でこの命令だけでは分岐先が特定できない場合、その直前のレジスタ設定命令を調べ、その値を調べるか、統計情報を採取する際のシミュレーション時に、分岐先が判明した時点で新たにセグメント分割位置を変更することにより構成要素の解析を行う。各関数と機械語プログラムのアドレスとの対応は、コンパイル時の情報を用いる。また、区間制約を与える際に使用するソースプログラムは別途与える。

【0073】(4) 前述の実施の形態では、命令列置換処理133において、アセンブラプログラムのレベルで置換を行い、変更されたアセンブラプログラムを再アセンブルすることにより再度、ハードウェア化部分も含めた性能解析を行う方法を示した。再度アセンブルするの

は、命令列置換処理により、プログラムのサイズが変わるため、分岐先のアドレス、データの格納アドレス等を再度設定しなおす必要があるからである。これに対し再アセンブルを行わず、内部データのみを修正する方法もある。内部データを修正する場合は、分岐命令やデータ参照等のメモリアドレスを使用している機械語命令の場所とそのアドレスに関する情報を別途保持し、プログラムサイズが変わった際に、機械語命令中のアドレス情報を直接変更できるようにする。また、構造/統計データ中のアドレスに関する情報を同時に更新する。

【0074】(5) 前述の実施の形態では、セグメント内の一部の命令列をハードウェアの起動命令列に置き換える例を示したが、複数のセグメントにまたがった命令列をハードウェアに置き換えることも可能である。この場合、構造/統計データのうちのセグメントの構成に関する情報も修正する必要がある。これは、修正のあつたセグメントに対し、それを含む関数の情報をたどり、逐次情報を修正することにより行う。

【0075】(6) 前述の実施の形態に示した支援方法では、そこで用いる処理や関連する処理が全て同じ計算機システム上で実行される。しかし、これらの処理は、異なるプロセッサ上に実現されていても、本発明の目的を損なうものではない。例えば、ハードウェアのシミュレーションに用いる論理シミュレータがその他の部分と異なるプロセッサ上で動作し、適宜必要な情報をプロセッサ間で通信することにより実現できる。また、入力とする組込みプログラムのコンパイル・アセンブルを他のプロセッサ上で行うこともできる。この場合、ファイルを通信あるいは記録媒体を介して転送するか、プロセッサ間でのファイル共有方法により共有を実現する。

【0076】
【発明の効果】本発明によれば、オブジェクトプログラムのシミュレーションの結果として、そのプログラムの各構成要素毎に処理時間に関連する情報を出力できるので、ユーザは、そのプログラムの中のハードウェア化すべき部分を各構成要素を手掛かりにして選択すればよいので、このハードウェア化すべき部分の選択しやすくなる。さらに、その際、シミュレーションを実行する区間を指定するようにすれば、ユーザが希望する区間にについて、上に述べたよう、構成要素を単位としてのシミュレーション結果が得られるので、ユーザが希望する部分の中からハードウェア化すべき部分をさらに絞ることが出来る。さらに、このシミュレーション区間の指定を、ソースプログラム上で指定する方法を採用した場合には、ユーザにとって、区間の指定が平易に出来る。

【0077】さらに、ユーザ指定の複数のシミュレーション区間にに対してシミュレーションを行い、その結果に基づいて、設計者が指定した処理時間等の条件を満たさない複数のシミュレーション区間を自動的に判別する場合には、設計者が希望する条件を満たさない区間の中か

らハードウェア化すべきプログラム部分をユーザは選択できるので、ハードウェア化すべき部分の選択が容易になる。

【0078】さらに、先行するシミュレーションの結果を利用して、ハードウェア化すべき部分を自動的に決め、その後シミュレーション中に使用するプログラムをそのハードウェア化する部分により自動的に更新する場合には、ユーザは、そのプログラムを自らが更新することなく、シミュレーションを繰り返すことが出来る。

【0079】

【図面の簡単な説明】

【図1】本発明によるハードウェア／ソフトウェア混在システムの設計支援装置の構成を示す図である。

【図2】区間制約指定処理のフローチャートである。

【図3】区間制約データを説明する図である。

【図4】プログラム構成要素解析処理のフローチャートである。

【図5】構造／統計データを説明する図である。

【図6】プログラム実行解析処理のフローチャートである。

【図7】統計解析処理のフローチャートである。

【図8】組み込みプログラムの例を説明する図である。

【図9】命令置換後のプログラムの例を示す図である。

【図10】プログラム実行トレース処理のフローチャートである。

【図11】区間制約違反部分に関して、処理時間に関する情報の表示動作を説明する図である。

【図12】区間制約違反の出力例である。

【図13】セグメント実行統計の出力例を示す図である。

【図14】セグメントの出力例を示す図である。

【図15】関数とセグメントの階層の出力例を示す図である。

【図16】関数実行統計の出力例を示す図である。

【図17】現在のセグメントの統計情報の更新処理のフローチャートである。

【図18】共通動作検索処理のフローチャートである。

【図19】共通動作命令列の例を示す図である。

【図20】ハードウェア生成処理のフローチャートである。

【図21】命令列置換処理のフローチャートである。

【図22】共通動作検索処理を説明するための図である。

【図23】共通動作検索処理で生成されるデータフローフラフの例を説明する図である。

【図24】命令列選択処理により選択された命令列を実行するハードウェアを例示する図である。

【図25】命令列選択処理により生成されるハードウェア記述言語の例を示す図である。

【図26】命令列選択処理により生成されるハードウェ

ア記述言語の他の例を示す図である。

【符号の説明】

801…プログラムの格納されるアドレス、802…命令コード、803…ラベル、804…プログラム部、805…コメント中に記録された高級言語のプログラム、806…アセンブリ言語、501…関数の構造データFN_struct、502…その関数の情報を表すFN(506)へのポインタfnp、503…リスト(518)へのポインタsg1、504…リスト(519)へのポインタprnt、505…リスト(520)へのポ

インタchd、506…関数の情報FN、507…関数の識別番号id、508…関数名(FN名)、509…関数の開始アドレスsa、510…関数の終了アドレスea、511…関数統計情報はその関数の統計情報、518…関数を構成するセグメントの集合を表すリスト、519…関数の上位階層の関数(その関数をコールする関数)のFN_structを指すポインタのリスト、520…関数の下位階層の関数のFN_structを指すポインタのリスト、512…セグメント間の関係を表すデータSG_struct、521…セグメント情報SG(525)へのポインタsgp、522…リスト541へのポインタpre、523…リスト542へのポインタsuc、524…このセグメントを含む関数へのポインタfnp、525…セグメント情報を表すデータSG、526…セグメントの識別番号id、527…セグメントの先頭アドレスにラベルがあった場合のラベル名、528…セグメントの開始アドレスsa、529…セグメントの終了アドレスea、530…セグメントの統計情報、541…このセグメントの直前に実行される

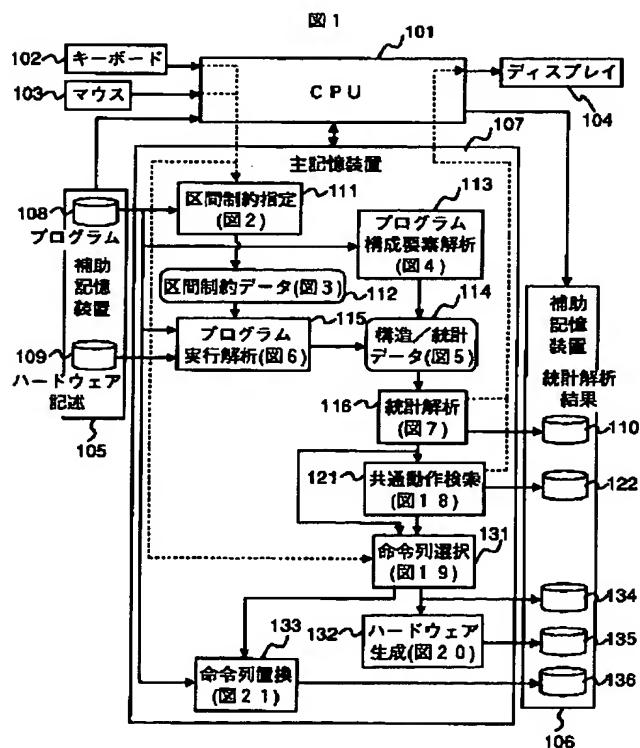
30 セグメントのリスト、542…このセグメントの直後に実行されるセグメントのリスト、1201…タイトル部分、1202…各行の番号#、1203…区間制約データの識別番号id(301)に対応する番号tsn#、1204…開始点アドレス(302)start、1205…終了点アドレス(303)end、1206…制約として与えられた処理時間(304)spec、1207…その区間の平均の処理時間ave、1208…その区間の実行回数cnt、1209…実際の処理時間が制約時間を超えた回数violate、1301…タイトル部分、1302…各行の番号#、1303…セグメント番号(SGのid 526)sg#、1304…セグメントの先頭アドレスのラベル(SGのlabel名527)label、1305…そのセグメントの開始アドレス(SGのsa 528)start、1306…終了アドレス(SGのea 529)end、1307…そのセグメントの合計処理時間total、1308…全処理時間に対するそのセグメントの処理時間の割合rate、1309…そのセグメントの実行回数call、1310…そのセグメントの平均処理時間ave、1311…最小処理時間min、1312…最大処

40 セグメントのリスト、542…このセグメントの直後に実行されるセグメントのリスト、1201…タイトル部分、1202…各行の番号#、1203…区間制約データの識別番号id(301)に対応する番号tsn#、1204…開始点アドレス(302)start、1205…終了点アドレス(303)end、1206…制約として与えられた処理時間(304)spec、1207…その区間の平均の処理時間ave、1208…その区間の実行回数cnt、1209…実際の処理時間が制約時間を超えた回数violate、1301…タイトル部分、1302…各行の番号#、1303…セグメント番号(SGのid 526)sg#、1304…セグメントの先頭アドレスのラベル(SGのlabel名527)label、1305…そのセグメントの開始アドレス(SGのsa 528)start、1306…終了アドレス(SGのea 529)end、1307…そのセグメントの合計処理時間total、1308…全処理時間に対するそのセグメントの処理時間の割合rate、1309…そのセグメントの実行回数call、1310…そのセグメントの平均処理時間ave、1311…最小処理時間min、1312…最大処

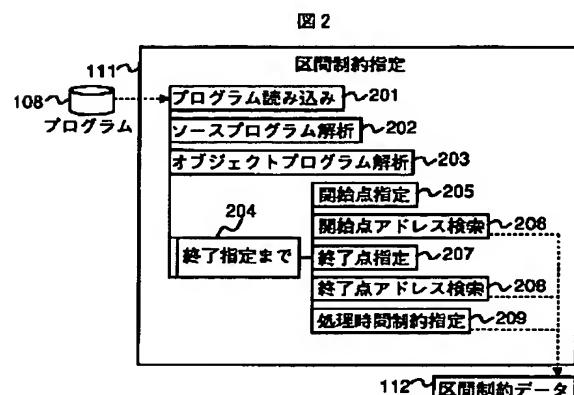
理時間max、1401…タイトル部、1402…アドレス部、1403…ラベル部、1404…命令部、1502・1503・1505…関数、1504・1506～1508…セグメント、1601…タイトル部、1602…各行の番号#、1603…関数の識別番号(FNのid関数507)fn#、1604…関数名(FNのFN名508)name、1605…開始アドレス(FNのsa509)start、1606…終了アドレス(FNのea510)end、1607…その関数自体の合計処理時間total、1608…全処理時間に対するその関数の処理時間の割合rate、1609…その関数の実行回数call、1610…その関数平均処理時間ave、1611…その関数以下の全処理時間

*理とその関数の処理時間合計a11、1612…a11の全処理時間に対する割合arate、2201～2205…入力ノード、2206～2211…ノード、2212～2215…クラスタ、1901…タイトル部、1902…各行の番号#、1903…共通動作の命令列のグループ番号gr#、1904…そのグループのコストの総和cost、1905…そのグループを構成するクラスタの種類の数#c1、1906…そのグループの共通動作命令列を含むセグメントの種類の数#sg、2301…入力ノード、2302…出力ノード、2303～2305…ノード、2401…アドレスバス、2402…データバス、2405～2408…入力レジスタ、2409～2411…演算器、2412…出力レジスタ。

【図1】



【図2】



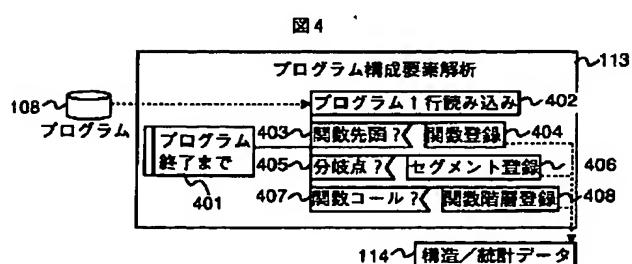
【図1.2】

図1.2							
#	tsn#	start	end	spec	ave	cnt	violate
1	5	0230	0254	8	13.6	100	100
2	2	0524	0580	800	983	1	1

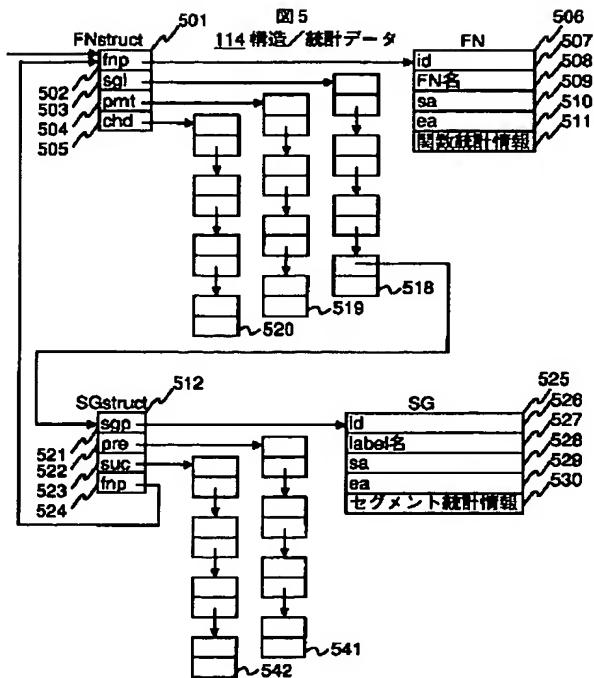
【図3】

図3 区間制約データ				
id	開始点アドレス	終了点アドレス	処理時間制約	統計情報
	301	302	303	304
				305

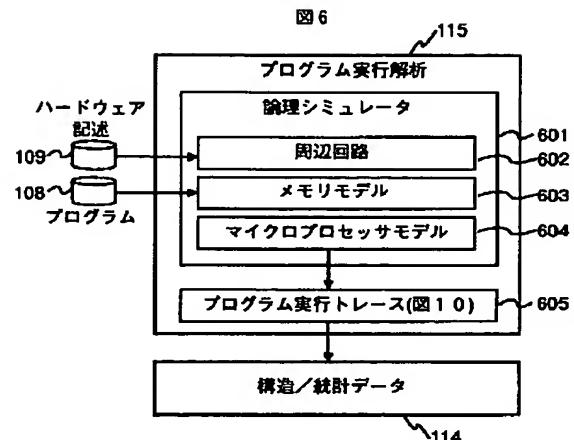
【図4】



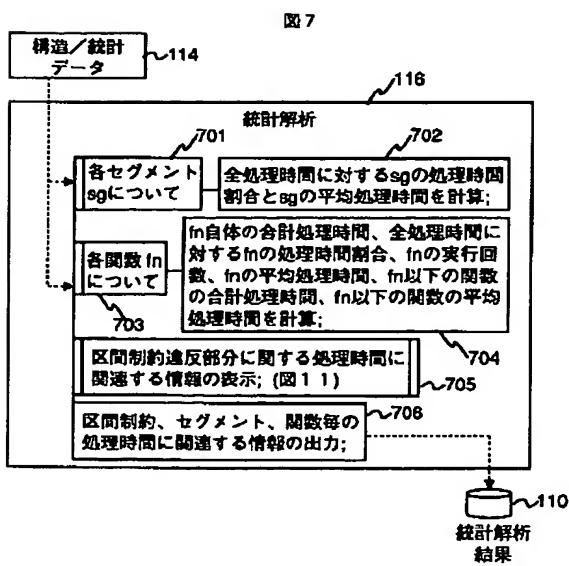
【図5】



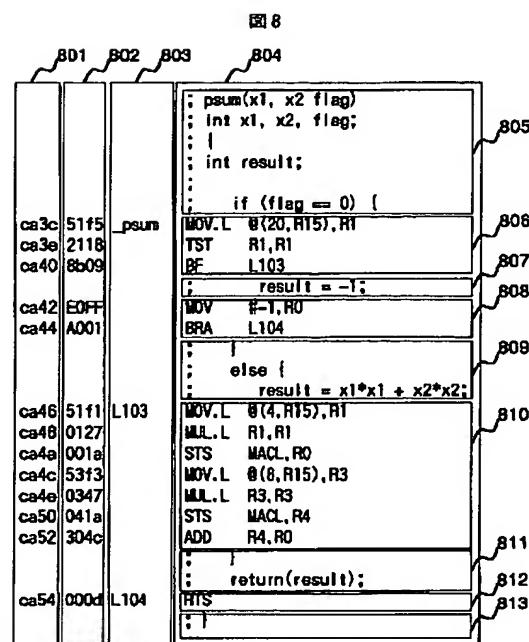
【図6】



【図7】



【図8】



【図 9】

図 9

```

: psum(x1, x2, flag)
: int x1, x2, flag;
: 
: int result;
: 
: if (flag == 0) {
_psum    MOV.L 0(12, R15), R1
        TST   R1, R1
        BF    L103
        result = -1;
        MOV   R1, R0
        BRA   L104
        ; 
: else {
        result = x1*x1 + x2*x2;
L103   MOV.L 0(4, R15), R1
        MOV.L R1, 0(0, R12)
        MOV.L 0(8, R15), R3
        MOV.L R3, 0(4, R12)
        MOV.L 0(12, R12), R0
        ; 
: return(result);
L104   RTS
: }

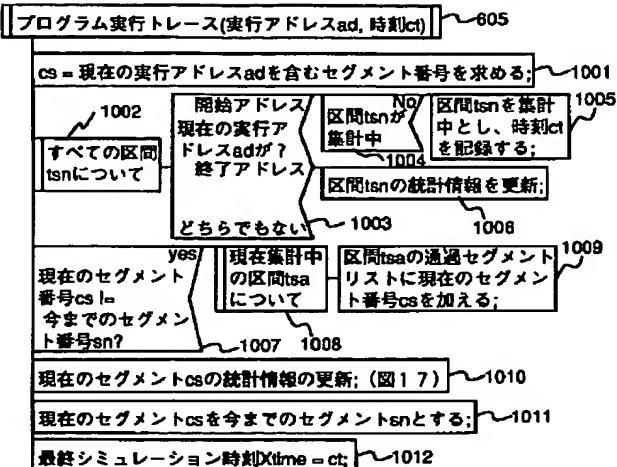
```

2701

2702

【図 10】

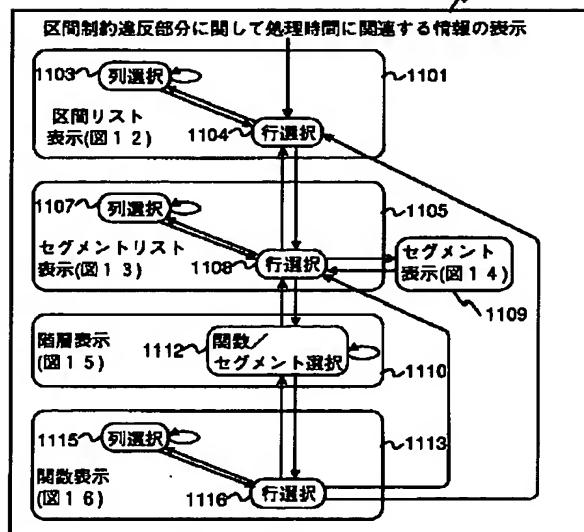
図 10



【図 13】

【図 11】

図 11



セグメント実行統計

#	sg#	label	start	end	total	rate	call	ave	min	max
1	13	L103	ca46	ca52	1358	10.2	100	13.6	13	15
2	34	fn1	c520	c618	1213	9.1	33	36.8	34	36
3	50	L352	83c0	842c	935	7.0	40	23.4	23	25
4	22		8430	84ec	885	6.5	20	43.3	43	45

1301 1302 1303 1304 1305 1306 1307 1308 1309 1310 1311 1312

【図 14】

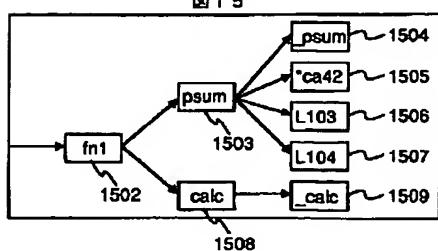
図 14

#1 sg#: 13 start:ca46 end:ca52	
ca46	L103
ca48	
ca49	
ca4c	
ca4e	
ca50	
ca52	

1402 1403 1404

【図 15】

図 15



1508

【図 16】

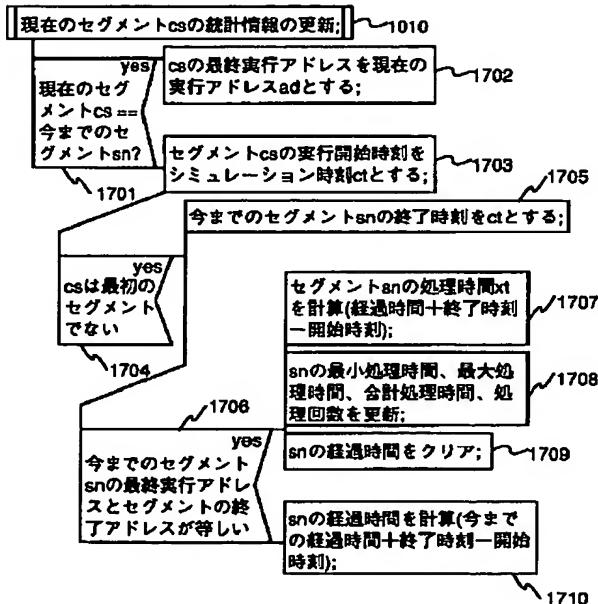
図 16

#	fn#	name	start	end	total	rate	call	ave	all	ave
1	13	fn1	c520	c618	1321	25.0	33	100.6	997	45.1
2	4	psum	c83c	c94c	1358	10.2	100	13.6	1358	10.2
3	16	calc	d610	d9fc	1320	9.9	26	59.8	1320	9.9
4	6	find	3154	34c4	1285	9.7	13	98.8	1285	9.7

1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612

【図17】

図17



【図19】

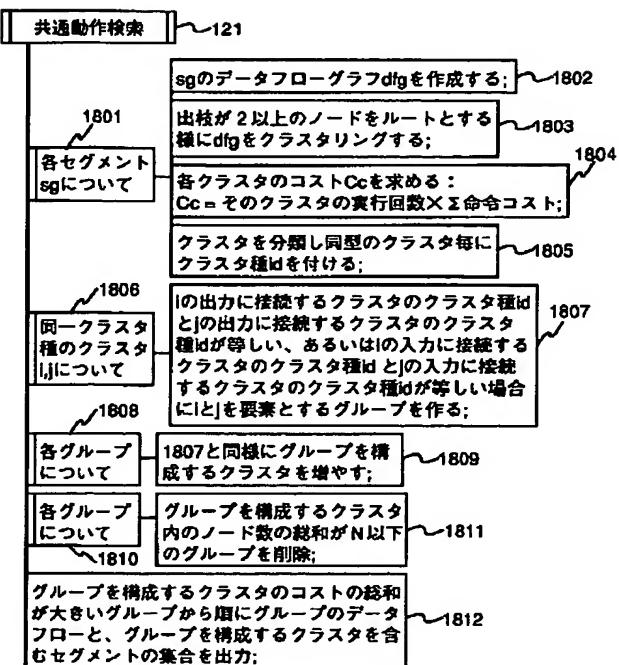
図19

共通動作命令列				
# gr#	cost	#cl	#sg	
1 5	1580	1	2	▲
2 2	524	2	3	▼

1901 1902 1903 1904 1905 1906

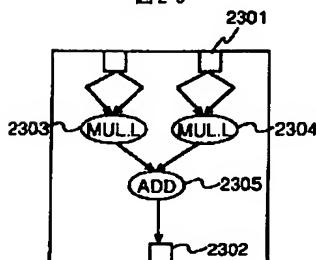
【図18】

図18



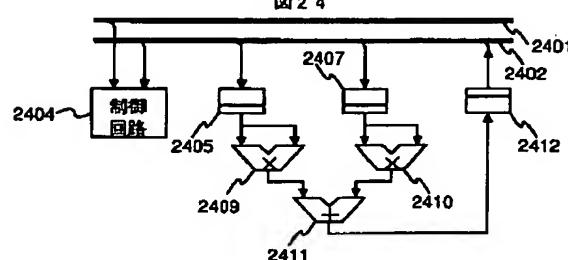
【図23】

図23

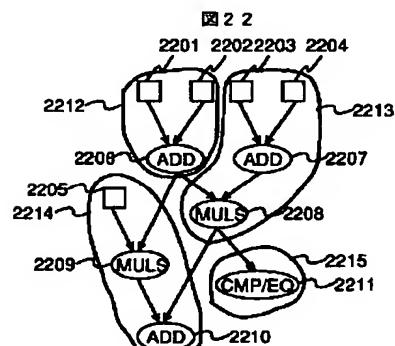


【図24】

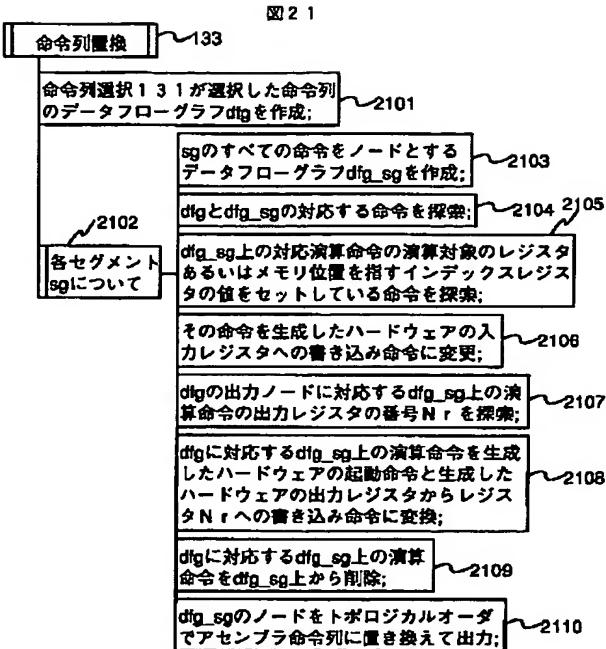
図24



【図22】



【図21】



【図25】

図25

```

entity psum_RTL is
  port ( ADR : in std_logic_vector(31 downto 0);
         DATA : inout std_logic_vector(31 downto 0);
         WR, CS, CLK : in std_logic;
         STRO, STR1 : in std_logic;
         );
end psum_RTL; 2501

architecture RTL of psum_RTL is
  2502
component REG32
  port ( DATA_IN : in std_logic_vector(31 downto 0);
         DATA_OUT : out std_logic_vector(31 downto 0);
         ST : in std_logic;
         );
end component;
component NULL
  port ( IN_A, IN_B : in std_logic_vector(31 downto 0);
         OUT_L, OUT_H : out std_logic_vector(31 downto 0);
         );
end component;
component ADD
  port ( IN_A, IN_C : in std_logic_vector(31 downto 0);
         OUT : out std_logic_vector(31 downto 0);
         COUT : out std_logic;
         );
end component;
constant psum_ADR: std_logic_vector(31 downto 0) := 制御アドレス;
constant psum_CYC: Integer := 実行サイクル数;
signal st_req_o : std_logic;
signal r_out0, r_out1 : std_logic_vector(31 downto 0);
signal op_out0, op_out1, op_out2 : std_logic_vector(31 downto 0);

```

続く

2503

【図26】

図26

```

begin
  process (RD, WR, CLK)
    variable STATE : integer := 0;
  begin
    if (CLK'event and CLK = '1') then
      st_req_o <= 0;
      if (STATE > 0) then
        STATE := STATE - 1;
        if (STATE = 0) then
          st_req_o <= '1';
        end if;
      end if;
    end if;
    if (ADR = psum_ADR) then
      if (WR'event and WR = '0' and CS = '0') then
        STATE := psum_CYC;
      end if;
    end if;
  end process; 2602
  2601
  I_REG_0: REG32 port map
    (DATA_IN => DATA, DATA_OUT => r_out0, ST => STRO );
  I_REG_1: REG32 port map
    (DATA_IN => DATA, DATA_OUT => r_out1, ST => STR1 );
  NULL_0: NULL port map
    (IN_A => r_out0, IN_B => r_out0, OUT_L => op_out0 );
  NULL_1: NULL port map
    (IN_A => r_out1, IN_B => r_out1, OUT_L => op_out1 );
  ADD_0: ADD port map
    (IN_A => op_out0, IN_B => op_out1, OUT => op_out2 );
  O_REG_0: REG32 port map
    (DATA_IN => op_out2, DATA_OUT => DATA, ST => st_req_o );

```